OrthoSecure Project Security policies --->

[1] First policy: Pod Security Policy (PSP)

A Pod Security Policy is a cluster-level resource that controls security-sensitive aspects of the Pod specification. The PSP we have defined in our project ORTHOSECURE includes several security constraints:

1. **Privileged**:
    - privileged: false - Pods cannot run in privileged mode.
2. **Allow Privilege Escalation**:
    - allowPrivilegeEscalation: false - Containers cannot allow privilege escalation.
3. **Required Drop Capabilities**:
    - requiredDropCapabilities: ["ALL"] - All Linux capabilities are dropped by default.
4. **Run As User**:
    - runAsUser.rule: MustRunAsNonRoot - Containers must run as a non-root user.
5. **SELinux**:
    - seLinux.rule: RunAsAny - Any SELinux context is allowed.
6. **File System Group**:
    - fsGroup.rule: MustRunAs - Enforces a specific range for file system group IDs.
7. **Supplemental Groups**:
    - supplementalGroups.rule: MustRunAs - Enforces a specific range for supplemental group IDs.
8. **Volumes**:
    - Only specific volume types are allowed (e.g., configMap, emptyDir, secret, persistentVolumeClaim).

[2] Second Policy: Gatekeeper ConstraintTemplate and Constraint

we have defined a `ConstraintTemplate` and a `Constraint` using Gatekeeper to enforce these policies:

1. **ConstraintTemplate**:

- Defines the structure and logic for the policy using Rego, a policy language.

- Specifies the conditions under which a violation occurs (e.g., privileged containers, privilege escalation, non-root user).

2. **Constraint**:

   - Applies the `ConstraintTemplate` to specific resources (e.g., Pods).

   - Matches the kinds of resources to which the policy should be applied.

How the Policy Works?

When you try to create a Pod, the following steps occur:

1. **Admission Controller**:

   - The Gatekeeper admission controller intercepts the request to create the Pod.

2. **Policy Evaluation**:

   - The admission controller evaluates the Pod against the defined `ConstraintTemplate` and `Constraint`.

3. **Violation Detection**:

   - If the Pod violates any of the specified constraints (e.g., running in privileged mode, allowing privilege escalation), the admission controller rejects the request.

4. **Error Message**:

   - The error message provides details about which constraints were violated.

Example of POD Rejection

Typical error when a user creates a pod without knowing our security policy

tayyab@Tayyab:/mnt/c/Users/Tayyab Qadri/OneDrive/Documents/OrthoSecure$ kubectl apply -f kubernetes/tests/test-pod-security.yaml

Error from server (Forbidden): error when creating "kubernetes/tests/test-pod-security.yaml": pods "test-pod" is forbidden: violates PodSecurity "restricted:latest": privileged (container "test-container" must not set securityContext.privileged=true), allowPrivilegeEscalation != false (container "test-container" must set securityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container "test-container" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or container "test-container" must set securityContext.runAsNonRoot=true), seccompProfile (pod or container "test-container" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost")

In this case, the Pod was rejected because it violated multiple constraints:

- `privileged (container "test-container" must not set securityContext.privileged=true)`

- `allowPrivilegeEscalation != false (container "test-container" must set securityContext.allowPrivilegeEscalation=false)`

- `unrestricted capabilities (container "test-container" must set securityContext.capabilities.drop=["ALL"])`

- `runAsNonRoot != true (pod or container "test-container" must set securityContext.runAsNonRoot=true)`

- `seccompProfile (pod or container "test-container" must set securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost")`

Compliant Pod Example

To create a compliant Pod, you need to ensure it adheres to the security constraints:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
  - name: test-container
    image: busybox
    command: ["sleep", "3600"]
    securityContext:
      privileged: false
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
```

This Pod should be accepted by the admission controller because it complies with the defined security policies.

[3] Third Policy: Role-Based Access Control (RBAC) and Namespace Segmentation

Enforce tight access controls to prevent unauthorized users from applying or modifying resources that violate security policies.

- **RBAC**: Assign users specific roles with the minimum permissions needed, ensuring they cannot bypass security policies.

For instance, allow users only to create resources in certain namespaces where security policies are enforced.

- Example RBAC Role to enforce restricted deployments:

```
yaml
Copy code
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: secure-namespace
  name: restricted-deployer
rules:
  - apiGroups: [""]
    resources: ["pods", "deployments"]
    verbs: ["create", "get", "list"]
```

- **Namespace segmentation**: Isolate workloads with different security needs into separate namespaces and enforce policies per namespace using **PodSecurity standards** or **Gatekeeper constraints**.

- **RBAC Role**: The `restricted-deployer` role ensures users or service accounts can only create, get, and list Pods and Deployments in the `secure-namespace`. They won't have the permissions to modify other resources.
- **RoleBinding**: The `RoleBinding` allows a specific user or service account to use the `restricted-deployer` role within the `secure-namespace`. This ensures that only authorized users can deploy resources in that namespace while adhering to security policies.

`kubernetes/rbac/rolebinding-restricted.yaml`

- **RoleBinding** that assigns the `restricted-deployer` role to a **ServiceAccount** (or user) within the **secure-namespace**.

```
yaml
Copy code
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: restricted-deployer-binding  # Name of the RoleBinding
  namespace: secure-namespace        # Namespace of the RoleBinding
subjects:
  - kind: ServiceAccount
    name: app-service-account        # Name of the ServiceAccount
    namespace: secure-namespace
roleRef:
  kind: Role
  name: restricted-deployer         # Role being bound
  apiGroup: rbac.authorization.k8s.io
```

Error message when security policy is not followed --- >

tayyab@Tayyab:/mnt/c/Users/Tayyab Qadri/OneDrive/Documents/OrthoSecure$ kubectl apply -f kubernetes/tests/test-rbac-pod.yaml

Error from server (Forbidden): error when creating "kubernetes/tests/test-rbac-pod.yaml": pods "test-pod" is forbidden: violates PodSecurity "restricted:latest": allowPrivilegeEscalation != false (container "test-container" must set securityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container "test-container" must set securityContext.capabilities.drop=["ALL"]; container "test-container" must not include "NET_ADMIN" in securityContext.capabilities.add), runAsNonRoot != true (pod or container "test-container" must set securityContext.runAsNonRoot=true), runAsUser=0 (container "test-container" must not set runAsUser=0), seccompProfile (container "test-container" must not set securityContext.seccompProfile.type to "Unconfined")

Yes, the error message confirms that your RBAC security policy is applied correctly. The Role and RoleBinding you have defined are restricting the actions that the app-service-account can perform within the secure-namespace.

Breakdown of the RBAC Configuration

- **Role** (restricted-deployer):
  - **Namespace**: secure-namespace
  - **Resources**: pods, deployments
  - **Verbs**: create, get, list
  - This role allows the service account to create, get, and list Pods and Deployments within the secure-namespace.
- **RoleBinding** (restricted-deployer-binding):
  - **Namespace**: secure-namespace
  - **Subjects**: app-service-account in the secure-namespace
  - **RoleRef**: Binds the restricted-deployer role to the app-service-account

Error Message Explanation

The error message you received when trying to create the Pod indicates that the Pod violates the Pod Security Policy. This means that the security constraints defined by the Pod Security Policy are being enforced correctly.

To create a compliant Pod, you need to adjust the Pod definition to meet the security constraints:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  namespace: secure-namespace
spec:
  containers:
```

```
- name: test-container
  image: busybox
  command: ["sleep", "3600"]
  securityContext:
    allowPrivilegeEscalation: false
    runAsNonRoot: true
    runAsUser: 1000  # Non-root user ID
    capabilities:
      drop: ["ALL"]
    seccompProfile:
      type: RuntimeDefault
```

Apply the updated Pod definition:

kubectl apply -f /c:/Users/Tayyab Qadri/OneDrive/Documents/OrthoSecure/kubernetes/tests/test-rbac-pod.yaml

This should create a Pod that complies with the Pod Security Policy and the RBAC constraints.

[4] Fourth policy: Pod Security Admission [secure namespace]

The labels pod-security.kubernetes.io/enforce, pod-security.kubernetes.io/audit, and pod-security.kubernetes.io/warn are part of Kubernetes' Pod Security Admission feature. These labels enforce, audit, and warn about Pod Security Standards (PSS) within the namespace.

Enforce: Prevents Pods that do not meet the specified security standards from being created.

Audit: Logs violations of the specified security standards.

Warn: Issues warnings for Pods that do not meet the specified security standards.

For Example

The restricted policy is the most restrictive Pod Security Standard. It enforces the following constraints:

- Containers must not run as privileged.
- Containers must not allow privilege escalation.
- Containers must drop all capabilities.
- Containers must run as a non-root user.
- Containers must set a seccomp profile to RuntimeDefault or Localhost.

Conclusion: By applying these labels to the secure-namespace, you are enforcing a strict security policy on all Pods within the namespace. This ensures that only Pods that comply with the restricted security standards can be created in this namespace.

To apply this namespace configuration, use the following command:

kubectl apply -f /c:/Users/Tayyab Qadri/OneDrive/Documents/OrthoSecure/kubernetes/policies/namespaces/secure-namespace.yaml

this will create the secure-namespace with the specifies security policies.

[5] Fifth policy: Network Policy Overview

A Network Policy in Kubernetes is used to control the traffic flow to and from Pods. It allows you to specify how groups of Pods are allowed to communicate with each other and other network endpoints.

Breakdown of the Network Policy

- **apiVersion**: networking.k8s.io/v1
  - Specifies the API version for the Network Policy resource.
- **kind**: NetworkPolicy
  - Indicates that this resource is a Network Policy.
- **metadata**:
  - **name**: allow-app-traffic
    - The name of the Network Policy.
  - **namespace**: default
    - The namespace where the Network Policy is applied.
- **spec**:
  - **podSelector**:
    - **matchLabels**:
    - app: my-app
    - This policy applies to Pods with the label app: my-app.
  - **policyTypes**:
    - Ingress
    - Controls incoming traffic to the selected Pods.
    - Egress
    - Controls outgoing traffic from the selected Pods.
  - **ingress**:
    - **from**:
      - **podSelector**:
    - **matchLabels**:
      - app: my-app
      - Allows incoming traffic only from Pods with the label app: my-app.
  - **ports**:
    - **protocol**: TCP
      - Allows TCP traffic.
    - **port**: 80
      - Allows traffic on port 80.
  - **egress**:
    - **to**:

        - **podSelector**:
        - **matchLabels**:
        - app: my-app
        - Allows outgoing traffic only to Pods with the label app: my-app.
    - **ports**:
        - **protocol**: TCP
        - Allows TCP traffic.
        - **port**: 80
        - Allows traffic on port 80.

Summary

This Network Policy named allow-app-traffic in the default namespace applies to Pods with the label app: my-app. It controls both incoming (ingress) and outgoing (egress) traffic:

Ingress: Allows incoming TCP traffic on port 80 only from Pods with the label app: my-app.

Egress: Allows outgoing TCP traffic on port 80 only to Pods with the label app: my-app.

This policy ensures that Pods with the label app: my-app can only communicate with each other on port 80 using TCP, both for incoming and outgoing traffic.

kubectl describe networkpolicy allow-app-traffic -n default

Name: allow-app-traffic

Namespace: default

Created on: 2024-12-29 12:57:41 +0000 UTC

Labels: <none>

Annotations: <none>

Spec:

PodSelector: app=my-app

Allowing ingress traffic:

To Port: 80/TCP

From:

PodSelector: app=my-app

Allowing egress traffic:

To Port: 80/TCP

To:

PodSelector: app=my-app

Policy Types: Ingress, Egress